The 15th Edition of the Multi-Agent Programming Contest - The GOAL-DTU Team

Alexander Birch Jensen, Jørgen Villadsen(✉), Jonas Weile, and Erik Kristian Gylling

Algorithms, Logic and Graphs Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Richard Petersens Plads, Building 324, 2800 Kongens Lyngby, Denmark
jovi@dtu.dk

Abstract. We provide an overview of the GOAL-DTU system for the Multi-Agent Programming Contest, including the overall strategy and how the system is designed to apply this strategy. Our agents are implemented using the GOAL programming language. We evaluate the performance of our agents in the contest and, finally, we discuss how to improve the system based on an analysis of its strengths and weaknesses.

1 Introduction

In 2020/2021 we participated as the GOAL-DTU team in the annual Multi-Agent Programming Contest (MAPC). We are using the GOAL agent programming language [1–4] and we are affiliated with the Technical University of Denmark (DTU). We participated in the contest in 2009 and 2010 as the Jason-DTU team [5,6], in 2011 and 2012 as the Python-DTU team [7,8], in 2013 and 2014 as the GOAL-DTU team [9], in 2015/2016 as the Python-DTU team [10], in 2017 and 2018 as the Jason-DTU team [11,12] and in 2019 as the GOAL-DTU team [13].

In 2020/2021 we had the *Agents Assemble II* scenario; this scenario expands upon the *Agents Assemble* scenario used in the 2019 contest. The *Agents Assemble II* scenario is a highly dynamic environment. The simulations used for the competition usually have a large number of agents that can move freely and even cause changes to the environment, which further adds to its complexity. As a new feature from the previous iteration, when an agent crosses the boundary of the map it will instantly reappear on the opposite side. This transition appears seamless to the agent and no triggers can be perceived. From the points of view of agents, the map may appear to be infinite while, in reality, all maps have finite dimensions. This means that agents may observe already known objects but consider them to be new knowledge. Most agents can usually be observed carrying blocks around the environment while clearing passages to enable their movement. Furthermore, random clear events may occur sporadically. As opposed to the clear actions of agents, which merely remove obstacles, the random clear events will both remove and add blocks in an area. Consequently, these random events can rapidly change the environment.

A key characteristic of our agent system is that agents share the same code base and knowledge. As such, the system has a single, universal type of agent. However, the agents still exhibit different behaviours at execution time, as the behaviour of an agent is determined by both its knowledge as well as its current beliefs and goals; these factors dictate the flow of the agent through logic rules and modules.

The paper is organized as follows:

- Section 2 covers the overall strategy of our agents.
- Section 3 describes the knowledge our agents acquire from the environment.
- Section 4 describes the movement of our agents.
- Section 5 describes how our agents communicate.
- Section 6 covers how our agents complete selected tasks.
- Section 7 evaluates our agents' performance in the contest.
- Section 8 discusses improvements to our agent system.
- Section 9 makes some concluding remarks.

We assume basic knowledge of the GOAL agent programming language [1–4]. Agents in GOAL are self-controlled independent entities, each interacting with the environment and communicating with other agents. The environment is continuously perceived to update each agent's mental state: its beliefs about the current state of affairs and its current goals. A mental state is implemented as a Prolog knowledge base. Rule-based decision-making enables each agent to continuously select an action based on its current mental state. GOAL advocates that agents are programmed to react to changes in their environment rather than executing predetermined plans. Such a reactive approach is not flawless either: it can be difficult for programmers to come up with logical rules that produce the desired behavior. However, overcoming this challenge often produces more flexible agents.

2 The Strategy of Our Agents

The main strategy of our agents is both proactive and reactive: agents explore the map to gather information and collect blocks while reacting to obstacles they meet along their way. Our agents employ an A* path finding algorithm to optimize short-term movement. Agents compose plans to solve available tasks that describe how patterns are to be aligned.

The logic of our strategy is implemented in GOAL in the so-called *main-module*. Here, the GOAL agent selects an action based on its knowledge, beliefs and goals, according to a prioritized list of predefined logic rules.

At the top-level, the strategy of agents is divided into two sub-strategies: exploration and task-solving. The only exception is a special task master agent that delegates tasks. This will be discussed in more detail in Sect. 6. We can consider the two sub-strategies to be the roots of a hierarchical goal network. Each of these goals branches into sub-goals that need to be achieved in order to complete the higher-level goal. Consider as an example a goal of solving a task. In order to achieve this goal, we first need to find and attach blocks needed for the task pattern before the pattern can be assembled and submitted. Each of these sub-tasks corresponds to sub-goals of the high-level goal of completing the task.

In the following section we will treat each of our two sub-strategies separately. We provide descriptions of the implemented agent logic below.

2.1 Exploration of the Map

Initially, all of our agents will act upon their initial goal to explore the map. This has two main purposes: to cover as much of the map as possible and to search for blocks or dispensers. When an agent locates a block or dispenser, it will attempt to attach two blocks on opposite sides; this in order to minimize the negative effect on maneuverability of carrying blocks. The mechanics of exploration is further explained in Sect. 4.

In the following, an action is selected based on the first applicable rule:

- If the agent does not have two blocks attached (which should be attached to opposite sides of the agent) and there is a block or dispenser within the field of vision of the agent:
  - If the agent is next to the block/dispenser, but it needs to rotate in order to attach a block:
    - If it is possible for the agent to rotate, then rotate.
    - If the agent can move in some direction, then move.
  - If the agent is next to a block, then attach the block.
  - If the agent is next to a dispenser, then request a block from that dispenser.
  - If the agent can move towards the block/dispenser, then move.
- If there is a good direction for the agent to move and explore, then move.
- If there are no other good options for the agent, then skip.

Note that agents sometimes move in a random direction if they are next to a block/dispenser and are required to rotate. The idea is that often just a few moves enable the agent to perform the rotation.

2.2 Accepting and Submitting Tasks

The win condition for the scenario is to score the most points. Points are gained by accepting tasks (at task boards) and submitting the required pattern. The currently available tasks are always perceivable by the agents. Based on the currently collected blocks, and information about potentially collected blocks (via block dispensers), we are able to compose so-called task plans. These delegate sub-tasks to each agent required to ultimately submit the pattern and score points.

Task plans are created by a single agent that has the task master role. A task plan delegates to the other agents the gathering and delivery of the required blocks. One of these agents is then selected to have the submit agent role: the agent that submits the pattern and completes the task once the pattern is assembled. All of this is explained more thoroughly in Sect. 6.1.

In the following scenarios, an action is selected based on the first applicable rule, if the current step in the task plan is to gather blocks required for a task:

- If the agent does not have room for the (maximum two) blocks we should deliver, then detach a block.
- If there is a dispenser or block of the required type within the agent's field of view:
    - If the agent is next to the object of interest, but it needs to rotate in order to attach the object:
        - If it is possible for the agent to rotate, then rotate.
        - If the agent can move in some direction, then move.
    - If the agent is next to a block, then attach the block.
    - If the agent is next to a dispenser, then request a block from that dispenser.
    - If the agent can move towards the object of interest, then move.
- If true then move towards the dispenser from the plan.

In the following scenarios, an action is selected based on the first applicable rule, if the current step in the task plan is to get to a task board:

- If the agent has a block attached that is not included in the plan and the agent needs space to attach a block included in the plan, then detach the block not included in the plan.
- If the agent is within a distance of two cells from the task board, then accept the task from the plan.
- If true then move towards the task board.

In the following scenarios, an action is selected based on the first applicable rule, if the current step in the task plan is to get to the specified goal cell and submit the task:

- If the agent is the submit agent:
    - If the pattern is blocked by an agent from the opposing team, try to perform a clear action on that agent.
    - If the agent is at the specified goal cell:
        - If the agent can submit the task, then submit the task.
        - If the agent can connect to a block in the task pattern, then connect to the block.
        - If the agent is in position and is simply waiting for other agents, and there are agents from the opposing team nearby, try to perform a clear action on one of them.
        - If the agent is in position, then skip.
        - If the agent needs to rotate, and it is possible, rotate.
    - If true, move towards the goal cell.
- If the agent is not the submit agent:

- If the agent has connected a block to another block, detach it.
- If a block/obstacle is blocking the pattern, then perform a clear action on it.
- If the agent is in position:
  - If the agent can connect a block to the task pattern, then connect the block.
  - If the agent is waiting for other agents:
    - If there are enemy agents nearby, then try to clear them.
    - If true then skip.
  - If rotation is required to connect a block to the task pattern, then rotate.
- If true, then move towards the goal cell.

3 Storing and Maintaining Information

In Sect. 2, we described how our agents operate based on a strategy that is implemented as decision rules. Such decisions are based on the information available to the agents: their mental states. Mental states of agents consist of their current beliefs and goals as well as a knowledge base which contains (static) domain-specific knowledge. All our agents have identical knowledge bases, as all agents are initiated with identical knowledge.

Our GOAL agents connect to the environment using the environment interface standard (EIS) [1]. This interface allows for communication between GOAL and the environment via text messages using the JSON (JavaScript Object Notation) format. A JSON message consists of a collection of name/value pairs. Generally speaking, we maintain the structure of perceived values when storing them as Prolog-like terms in the belief base. In some cases, we expand with additional information that is not made available by the environment, such as the agent's own attached block which can be inferred by the agents.

Because the *Agents Assemble II* scenario features a highly dynamic environment, our agents do not rely on creating a complete representation of their environment—it is almost impossible to maintain an accurate picture. Instead, our agents mostly rely on their current percepts. Agents only perceive objects, including other agents, within a limited field of vision. This field of vision corresponds to a circle in the taxicab geometry, where the radius for the circle is given by the environment, see Fig. 1.

3.1 Immutable Objects in the Environment

The highly dynamic environment implies that information perceived by agents may become outdated quickly. However, there are a number of immutable objects in the environment, namely the block dispensers, goal cells, and task boards. All of these are unaffected by clear actions and are static throughout a simulation. Our agents will therefore permanently store all known positions of these objects in their belief bases. The fact that dispensers, goal cells, and task boards are immutable facilitates our task planning algorithm which is explored further in Sect. 6.1.

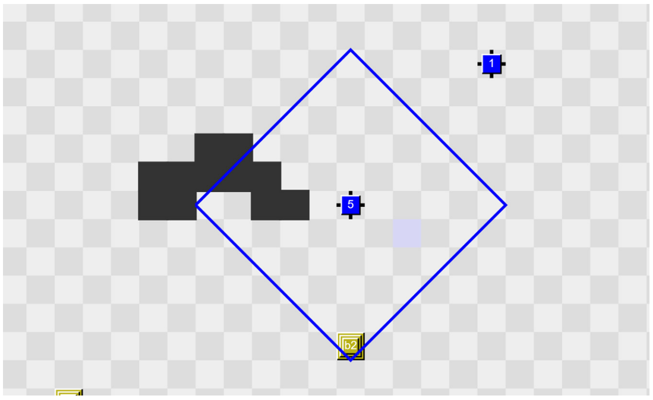

Fig. 1. This figure depicts two agents. The agent in the center of the picture (labeled 5) is surrounded by a blue diamond shape. This shape is a circle of radius five in the taxicab geometry, and it represents the field of vision of the agent. The agent 5 can thus only percept objects within the blue shape, therefore, the two agents in the picture cannot perceive one another. The picture is taken from the monitor that has been developed for the contest. (Color figure online)

Because these immutable objects are static throughout a simulation, we are able to store and share this information amongst our agents without worrying about the information becoming outdated at a later point. The details of agent communication are covered in Sect. 5. In fact, our decentralized planning is only enabled because of our information sharing strategy.

3.2 Agreeing on Coordinates

Each of our agents maintain its own separate coordinate system. The origin of this coordinate system is initialized as the agent's starting position. The separate coordinate systems pose a challenge when agents share information that is relative to their own coordinates. One solution to this problem would be to have agents agree on a single coordinate system. This implies that whenever agents meet and share their relative coordinates, they should decide on a common coordinate system. However, we found that this approach requires extensive protocols to be developed in order to ensure connected agents always agree on the same coordinate system. We have instead employed a solution in which each agent stores the offsets of all other agents. This means that whenever an agent receives information from another agent, the coordinates are easily translated. One drawback to our approach is that agents need to maintain a synchronized data set of offsets amongst all agents to avoid false information (such issues may in turn propagate throughout the execution and cause beliefs to become increasingly out of sync). How agents meet and share their coordinates is explained more thoroughly in Sect. 5.1.

3.3 Inferring the Map Dimensions

We previously described how, compared to the scenario of the 2019 contest, the new scenario allows movement across the boundaries of the map. This poses a challenge in terms of inferring the correct map dimensions and avoiding a misrepresentation of the map. Another challenge posed by this is in relation to path finding where an agent may select sub-optimal routes due to a lack of knowledge, see Fig. 2.

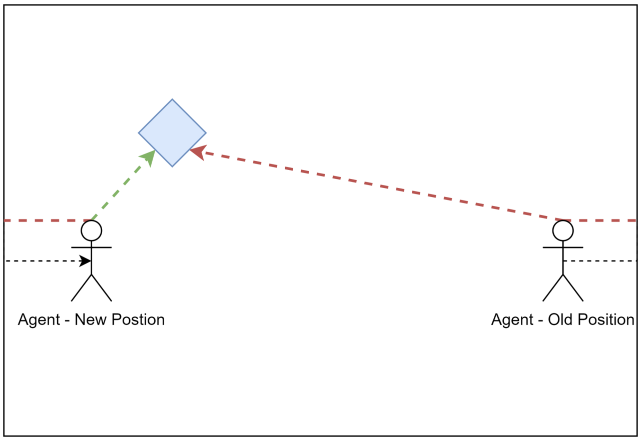

Fig. 2. Here, an agent unknowingly moves across the border of the map and reappears on the other side, represented by the black arrow. Once at the other the side, the agent has to reach the resource represented by the blue rhombus. The agent falsely believes, that the fastest way to reach this resource is by crossing the map again, and will then traverse the entire map. This path is represented by the dashed red line. In reality, the agent is very close to the resource, and the shortest path is shown by the dashed green line. (Color figure online)

To avoid the mentioned misrepresentation of the map, agents employ a strategy that intends to discover, and communicate to other agents, the dimensions of the map. This plays a central role in estimating distances. If the dimensions of the map are inferred, agents may knowingly decide to cross the edges of the map when this is deemed beneficial. Thus, knowing the dimensions of the map not only ensures that the agents do not flood their belief bases with false (or rather, sub-optimal) locations of resources, but also optimizes the agent's movement.

The dimensions of the map are inferred as an agent crosses the map and reappears on the other side, unknowingly. When the agent meets another agent, to which it was connected to prior to crossing the map, it infers the dimensions from the stored agent offsets. If their shared information does not match, it can be deduced that one of the agents must have crossed the map. The map dimensions can then be inferred from this discrepancy and are shared with all other agents.

4 Moving About in the Environment

In the following section, we touch upon the various movement strategies that are employed by our agents, depending on their current goals. The use of movement strategies can generally be divided into two cases: one for general map exploration and one for movement towards a fixed position.

4.1 General Map Exploration

As previously described, all agents are initialized with the goal of exploring the map. Here, an agent's sole purpose is to reach unexplored regions of the map. The efficiency with which any single agent reaches these unexplored regions is considered of low importance. However, we note that a more effective, collective exploration could benefit our system by allowing for a quicker transition to task-solving.

Our design philosophy is that exploration should have a relatively low computational complexity, considering all agents are exploring initially. Furthermore, from last year's competition, we learned that a somewhat random approach to exploration is adequate. We employ a simple pseudo-random heuristic, defined in Eq. 1. The heuristic is based on expected benefits of moving in each possible direction and heavily favors cells that are unexplored. Here, d is the direction being evaluated, ΔS is the number of steps since the cell was visited, and $|\Delta x(d)| + |\Delta y(d)|$ is the Manhattan distance to the cell from the current location of the agent. The direction with the best value is then chosen.

$$h(d) = \sum_{visited} \begin{cases} \dfrac{|\Delta x(d)| + |\Delta y(d)|}{\Delta S^2} & \text{if } |\Delta x(d)| + |\Delta y(d)| \leq 30 \text{ and } \Delta S > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Our current implementation does not enable agents to share information about visited cells. We note that doing so could potentially lead to faster and more thorough exploration. This could be particularly beneficial on larger maps.

4.2 Movement Towards a Fixed Position

As explained in Sect. 3, the agents do not maintain a complete representation of the map. Because of this, we cannot rely on classical route planning algorithms. In the previous year's contest, our agents employed a simple heuristic similar to the exploration heuristic to determine a movement in each simulation step. In terms of computational complexity, this is a lightweight solution compared to heavy route planning algorithms and it is easy to implement. However, this naive approach proved to be inefficient in cases where the agent encounters obstacles. We observed that agents would spend a lot of steps trying to get around obstacles and progress was often severely impacted by agents getting stuck.

To circumvent this issue, our current iteration of the system employs a solution that mixes the heuristics-based approach with local route planning. The idea is to get the best of both worlds. In case the agent is moving towards a fixed location, it will thus first generate a move based on the simple movement heuristics. In case the move is constructive (see below), it will simply perform it. Otherwise, the agent will instead resort to route planning within its field of vision. The optimal position in the field of vision, with regard to the target location, is set as a *way point*. The path planning returns a sequence of actions that will enable the agent to reach this way point. The agent will keep progressing through the sequence as long as possible. In case the sequence becomes invalid or the agent reaches its way point, the process starts over: a way point is found, and the agent attempts to generate an action with the simple heuristic, otherwise resorting to route planning. Once the fixed location is within the agent's field of vision, this location will be used as the final way point.

Any move that brings the agent strictly closer to the target position, while avoiding recently visited positions, is considered constructive. We consider a position to be recently visited if it was visited fewer than 3 steps ago. This somewhat remedies situations where the agent might otherwise get caught in dead ends, endlessly moving back and forth without making progress.

Our definition of constructive moves is quite restrictive. For instance, whenever an agent has to go around an obstacle, the move will always be considered nonconstructive. To get the full benefits of the heuristics-based approach, the definition of constructive moves is relaxed. The relaxation involves categorizing all moves immediately next to obstacles as constructive as well. More precisely, if an agent moves along an obstacle that is closer to the target position than the agent, the move is constructive. This means that the agent can move around obstacles without resorting to route planning. However, this may also potentially lead to undesirable situations, as was observed during the competition: if two agents try to move past each other, they could end up moving further away from their respective target positions than desired. This specific problem is illustrated in Fig. 3. This observed problem has us questioning whether the described relaxation has been beneficial or not.

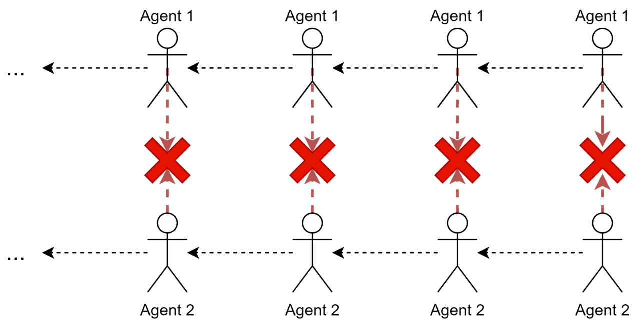

Fig. 3. This figure depicts two agents that wish to pass each other. Agent 2 wishes to go north, and agent 1 wishes to go south. This is not possible though, as the agents are in the way of each other. According to the relaxed heuristic, both agents can move west though. But this will simply bring the agents to the same dilemma once more.

In contrast to last year's implementation, we are now also utilizing the *clear* action with the introduction of local route planning. We expect that agents should no longer get stuck. We also expect general improvements to the efficiency of agent movement.

Route planning is implemented using the A* path finding algorithm. The heuristic used for the A* algorithm is simply the sum of the step cost and the distance to the target position. The search is terminated once a position at the edge of the field of vision is chosen, or if the target position is reached. To ensure effective route planning, the A* algorithm has been implemented in 2 versions. One version does not consider clear actions, and one does. This is due to the extra complexity introduced by the clear action. It was experimentally found that in a lot of cases, searches including clear actions resulted in drastic impacts to the running time. Consequently, our agents first initialize a search without enabling clearing of obstacles. If this is not feasible, the agent skips the current step to focus resources on computing a path that relies on clear actions as well.

5 Communication Between Agents

Solving any task in unison requires communication among agents. Agents need to agree on a task to solve as well as a goal cell at which to deliver the task. They also need to assign the collection and delivery of blocks required to solve it. The challenge is that each agent has its own coordinate system and thus agents cannot communicate about locations on a global level. Therefore, agents must somehow merge their coordinate systems, and this process is what we describe as connecting to other agents. The actual communication itself is facilitated by the extensive communication scheme that is provided by the GOAL agent programming language. The communication features of the GOAL programming language facilitate communication between agents through direct messaging or using channels. In general, the philosophy is to form communication channels instead of direct messaging when multiple agents are involved. As an example, our agents communicate via a channel *explore*, which is created during initialization of the system, where they share information to infer the position of other agents.

5.1 Connecting to Other Agents

For our agents, establishing connections equates to learning the origins of the other agents, thus allowing for translation of coordinates between coordinate systems. As a result, it enables agents to communicate the locations of resources. Agents may establish a connection when they are inside each other's field of vision. Since agents do not know the identities of other agents they encounter, the agents compare other perceivable objects in their (assumed) shared field of vision. This is achieved by the agents broadcasting their current position along with coordinates of the objects within their field of vision. In case another agent identifies the same objects, including the broadcasting agent, the two agents will establish a connection. Once agents are connected, they will start sharing information. Note that agents communicate coordinates relative to their starting position. The agent receiving the information will use stored knowledge about the offset between their starting positions to allow for an efficient translation between coordinates.

Ensuring the Correctness of Connections
It is important to ensure that agents will not mistakenly establish connections with agents which were not encountered. This may happen in rare cases where multiple agents encounter each other with similar objects in the shared field of vision. To remedy this, agents check whether there are multiple identical broadcasts, in which case no connections are established. The fact that multiple agents have similar broadcasts indicates that it will not be possible to distinguish those agents from one another.

A further challenge is imposed by the fact that in GOAL, the delivery of messages is not guaranteed in a certain step; it can merely be assumed that all messages will be delivered eventually. As such, the delay in receiving a message might be longer than a single step which can lead to situations where an agent has not received all broadcasts, and thus cannot ensure that no identical broadcasts are present. In our current implementation, all agents are required to broadcast, even if there are no other agents within its field of vision, in which case an empty broadcast is sent. Thus an agent will have to wait until it has received broadcasts from all other agents, before it establishes any new connections. This means, that there is a delay between encountering agents and establishing connections to these agents.

Connection Networks
Up to this point, we have not yet considered cascading effects when establishing connections. If agents can only connect to other agents within their field of vision, all agents have to encounter each other in order to establish connections between all agents. This is obviously not very efficient, especially for larger maps. In many cases, the system might never reach such a state within the limited time frame of a simulation. We are interested in optimizing this behaviour as our task planning relies on connections being established between agents. Essentially, having connections to other agents simply boils down to agents knowing the starting positions of other agents relative to their own.

To explain the process of connection cascading, consider the situation depicted in Fig. 4. The situation has three different networks of connected agents: the first network is made up of Agent 1, 2 and 3, the second network is made up of Agent 4 and 5, and the final network is just Agent 6. A network of connected agents is a set of agents, each knowing the offset of all other agents in the network. In the depicted situation, encounters between Agent 6 and 3, and Agent 4 and 2 occur simultaneously. As such, there is a possibility to connect the three networks. The challenge is now to ensure that all agents in each network establish connections to all agents of the two other networks. In general, this is achieved by having agents broadcast their connections. This allows an agent to infer the resulting network by combining all new connections and

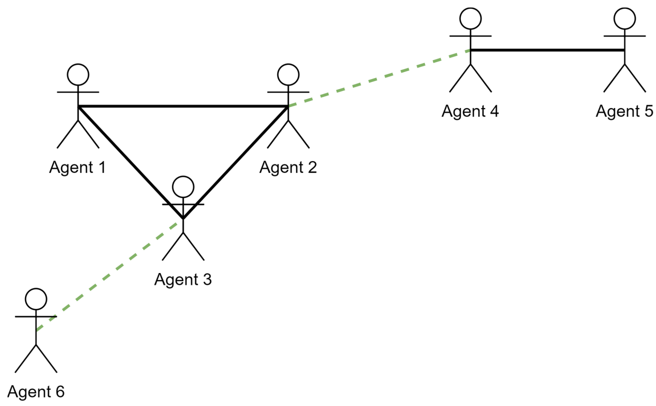

Fig. 4. The figure depicts three existing connection networks symbolised by the solid black lines—one network is the singular agent 6. Two new connections are established: a connection between agent 3 and agent 6, and a connection between agent 2 and agent 4. The resulting network then connects all 6 agents, ensuring that all agents know the offset of all other agents.

existing networks. To optimize the process, only a single agent of each network will compute the resulting network. Once all information necessary to establish new connections has been gathered, said agent will share this information with its own network.

6 Constructing and Executing Task Plans

This section covers how task plans are constructed by the task master and executed by the other agents.

A single agent is dynamically assigned to be *task master*. This agent is responsible for creating task plans and assigning agents to different tasks. The choice of having a single task master is to simplify resource assignment. With just a single agent responsible for computing task plans, we avoid having to deal with mechanisms for ensuring that resources are not allocated for multiple plans by multiple agents—a thread with a single pool of resources makes this much easier to achieve. The main principle of our dynamic task master assignment is to ensure that the chosen task master is connected to the majority of the agents. This is done by delaying the task master assignment to when at least half of the agents in the simulation have connected to each other, as described in Sect. 5.1. Outside of the mentioned delay, we find that the only drawback to our centralisation of task planning is the computational complexity. However, it is not clear if and how this could potentially be improved by parallelization.

The task master keeps track of goal cells and agents that are allocated by the currently active task plans. All cells of a goal zone are considered to be allocated if an active task plan instructs agents to deliver the task in the said goal zone.

The intent is to prevent agents from interfering with each other when solving tasks. However, we note that our current implementation is overly restrictive. One possible solution would be to allow for goal zones to be divided into subareas that may then be allocated individually.

6.1 Construction of Task Plans

Our task planning is initialized by the task master, which orders all the tasks that do not have an active task plan. Tasks are ordered by their reward and deadline, and the most promising task is considered first. The task master will ask all other agents in its network for their current attachments and beliefs about the locations of various block dispensers. The agents will respond with their beliefs about the resources needed for the task, whether the agent knows the location of a task board, and finally an estimated delivery time. The task master collects all responses to see if it is possible to compute a plan. A task describes a specific pattern to complete, and based on the combined resources of the agents, it is possible to search for an assignment of agents to blocks in the pattern which also considers the constraints of the scenario, i.e. how blocks can be attached as well as the positioning of agents for assembling the pattern and submitting it. This assignment delegates specific blocks of the pattern to specific agents and this cannot be changed without dropping the plan completely. While this limitation makes the solution less flexible, we find that the greatest hurdle is to have the agents reach a point where the pattern can be assembled. In case it is not possible to compute a plan, the task master will consider alternative tasks following the task order. In case a plan is created, each agent receives a version of the task plan that is specifically tailored to its perspective: which part of the pattern it should provide, how it should position itself for the assembly, etc.

The task plans delegate the task of submitting to one agent. In doing so, the submitting agent is also instructed to submit the task at a specific goal cell. While the approach has a few drawbacks, there are also several advantages: each agent knows exactly where to deliver its block and agents can easily identify blocks and obstacles to clear that will otherwise obstruct the assembly of the pattern. One drawback is that agents of opposing teams may obstruct the assembly. In this case, our agents will try to perform clear actions on those agents, hopefully leading them to move, but this is not guaranteed to succeed. A better solution would likely be to have a more dynamic approach that uses a fixed goal cell when possible, but with the ability to recompute a new goal cell if needed. As of now, our agents simply give up submitting the task after some time if they fail to make progress.

6.2 Execution of Task Plans

As described above, agents assigned to a task receive a task plan that is local to their perspective. For simplicity, an agent will only be assigned to deliver blocks of a single type. Once assigned to a task, each agent sets out to gather the blocks it is assigned to deliver. The submit agent is also tasked with accepting the task from a task board, as well as submitting the task.

Our agents continuously re-evaluate whether it is still possible to complete the task. For instance, it may happen that an agent loses a block that was to be delivered due to a clear action or event. In such a case, the agent checks if it is possible to re-acquire a block of the required type and deliver it within the deadline. If this is not possible, it will broadcast to the task channel that the task plan should be dropped, at which point all agents are unassigned from the task.

Once the agents are assembled, they will build the pattern outwards from the submit agent. Agents will connect one at a time to the submit agent and then release the corresponding block. If the agent has no more blocks to deliver, it will consider its task as fulfilled and be available to accept other tasks. When the pattern is completed the submit agent will then submit it.

7 Evaluation of Matches

GOAL-DTU competed in four matches against four different opponents in the Multi-Agent Programming Contest of 2020/2021. A match consisted of three simulations, and a simulation started automatically when the previous simulation finished. GOAL-DTU took part in two additional simulations after the contest. In these two simulations, all teams competed against each other.

7.1 GOAL-DTU vs. LTI-USP

LTI-USP allows multiple agents to accept the same task, such that multiple groups of agents could work on the same task. For each task, GOAL-DTU allows only one agent to accept it and only one group of agents to assemble it. Having multiple groups of agents to work on the same task didn't make LTI-USP's agents more effective. It only made their points per resource ratio smaller. It was not the best approach in the second simulation because agents of GOAL-DTU were consistently faster in assembling patterns than LTI-USP's agents. It is shown in Fig. 13 that GOAL-DTU completed more tasks, but it does not show how many of those tasks LTI-USP were trying to complete (Figs. 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18 and 19).

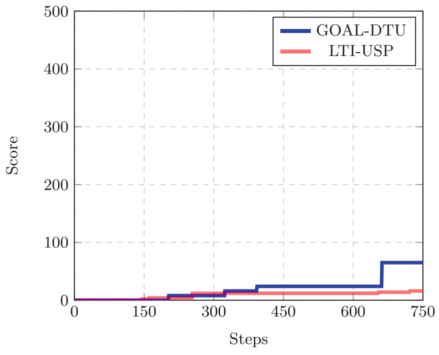

Fig. 5. Score: LTI-USP (1)

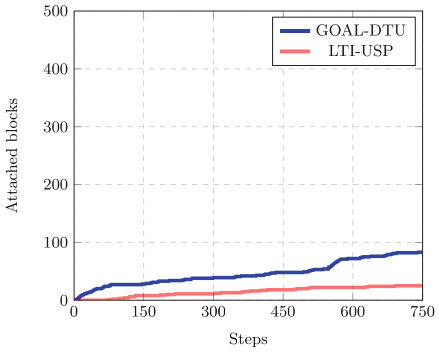

Fig. 6. Blocks: LTI-USP (1)

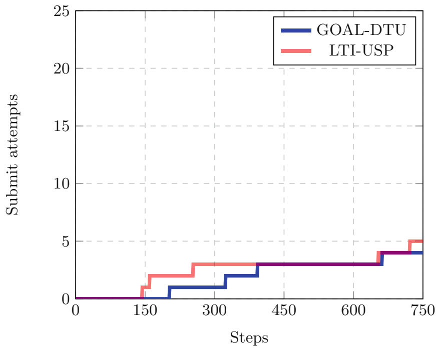

Fig. 7. Submits: LTI-USP (1)

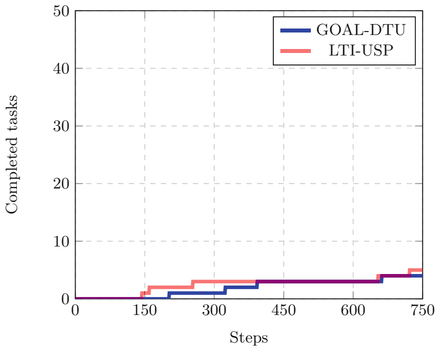

Fig. 8. Tasks: LTI-USP (1)

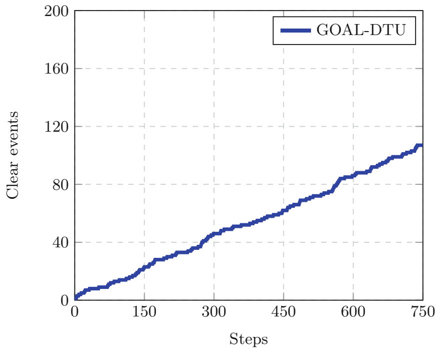

Fig. 9. Clear: LTI-USP (1)

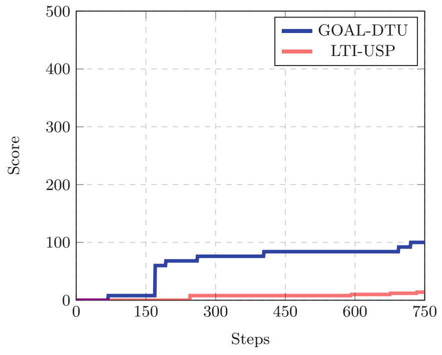

Fig. 10. Score: LTI-USP (2)

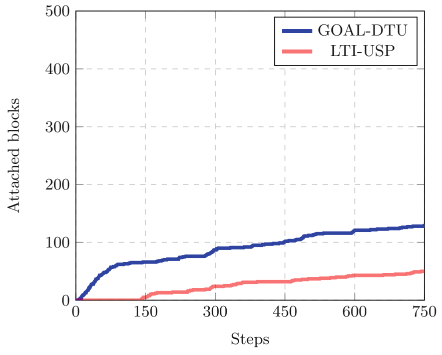

Fig. 11. Blocks: LTI-USP (2)

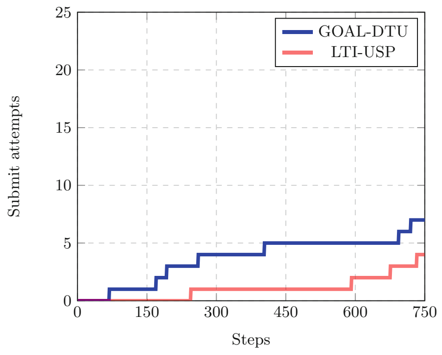

Fig. 12. Submits: LTI-USP (2)

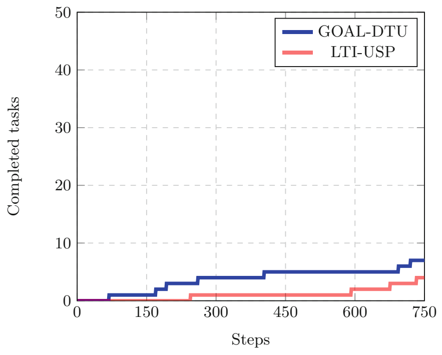

Fig. 13. Tasks: LTI-USP (2)

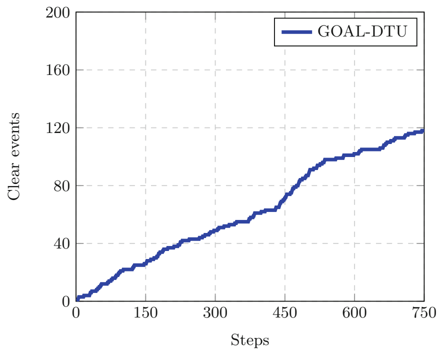

Fig. 14. Clear: LTI-USP (2)

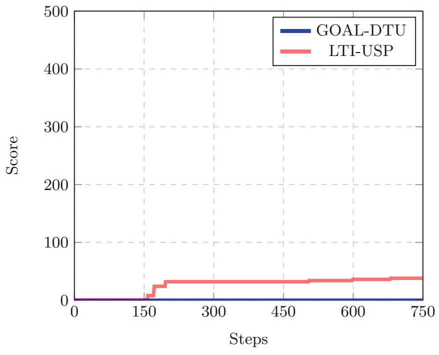

Fig. 15. Score: LTI-USP (3)

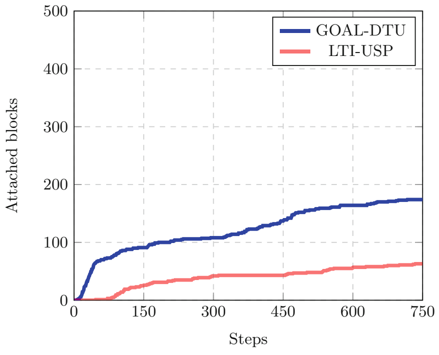

Fig. 16. Blocks: LTI-USP (3)

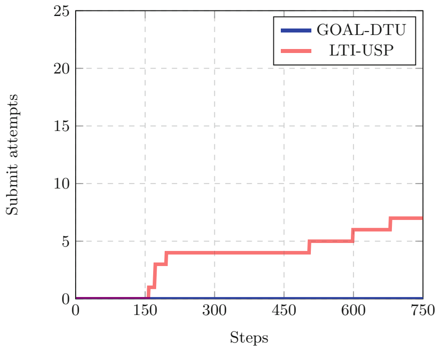

Fig. 17. Submits: LTI-USP (3)

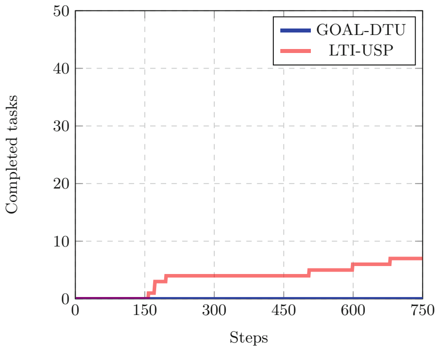

Fig. 18. Tasks: LTI-USP (3)

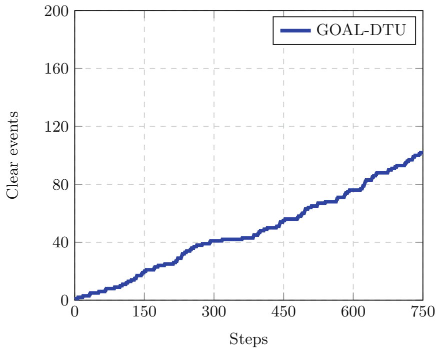

Fig. 19. Clear: LTI-USP (3)

GOAL-DTU had some problems throughout the simulations. The agents moved in each other's way, and sometimes clusters formed. We suspect a minor bug in the explore algorithm is causing agents to move towards each other. The intention is for agents not to interfere with the movement of other nearby agents. This resulted in problems in goal zones where agents gave up on patterns that were very close to being assembled. When giving up on tasks, the agents should detach blocks and move on with only two blocks. This was not always the case, since agents didn't detach indirectly connected blocks. These agents didn't seem to be aware of the indirectly connected blocks when moving around afterward. They tried to perform move actions that weren't applicable with the indirectly connected blocks.

7.2 GOAL-DTU vs. MLFC

We lost no matches against MLFC. This was also by fortune, as the second match was a 0-0 draw.

MLFC spent a lot of resources clearing our agents, with some, albeit limited, success. They did manage to clear GOAL-DTU agents waiting in the goal zones on more than one occasion, which resulted in the GOAL-DTU agents dropping their tasks. Our agents were not defending themselves as well as we intended. This also explains why MLFC managed several critical clears.

As seen in Fig. 38 and Fig. 48, we solve a substantially higher number of tasks than MLFC in the first and third simulation. But the one task that MLFC solves in the third simulation has a very high reward.

In both the second and the third simulation we experience some technical issues, although the issues must be described as more severe in the second simulation. To our luck, MLFC experience equally severe issues in the second simulation, and we manage to get a draw. MLFC is therefore the only team we did not lose a match against (Figs. 35, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47 and 49).

7.3 GOAL-DTU vs. FIT-BUT

FIT-BUT won two of the three simulations. Thus, FIT-BUT is the only team against whom we won just a single match. Clearly, FIT-BUT has developed a strong and effective system, and they manage to submit a very high number of tasks. Even in the first simulation, which we won, FIT-BUT manages to solve more tasks than we do, see Fig. 23. However, the tasks we solve have a higher reward, either due to increased complexity, or because we manage to solve them quite fast (Figs. 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33 and 34).

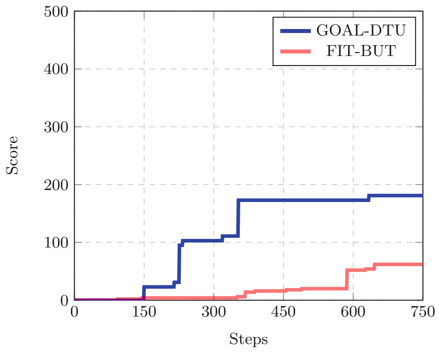

Fig. 20. Score: FIT-BUT (1)

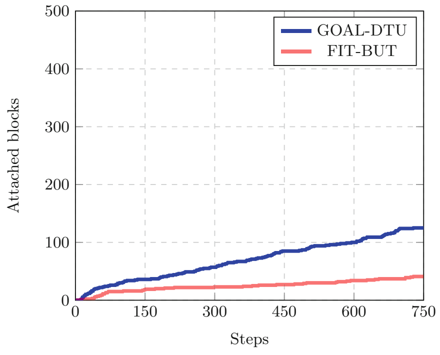

Fig. 21. Blocks: FIT-BUT (1)

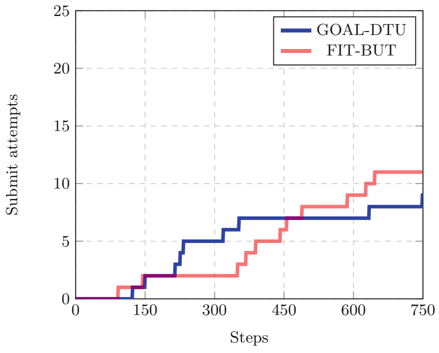

Fig. 22. Submits: FIT-BUT (1)

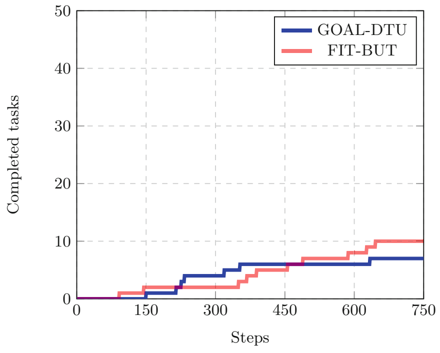

Fig. 23. Tasks: FIT-BUT (1)

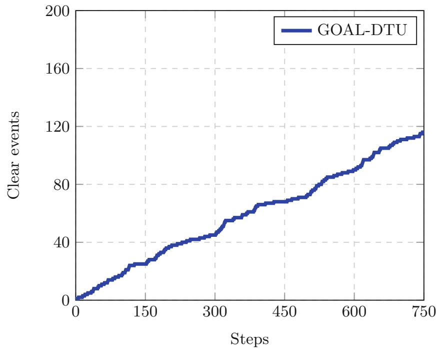

Fig. 24. Clear: FIT-BUT (1)

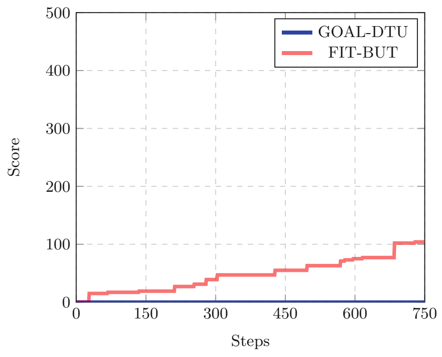

Fig. 25. Score: FIT-BUT (2)

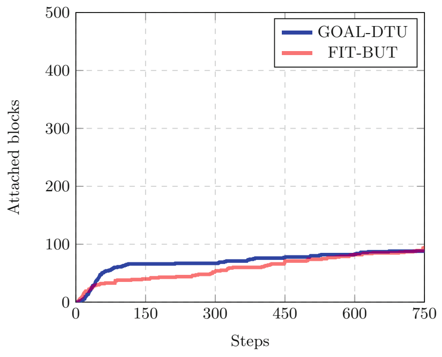

Fig. 26. Blocks: FIT-BUT (2)

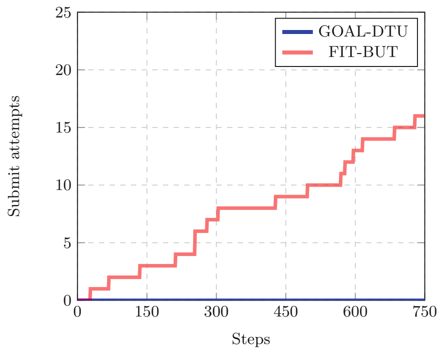

Fig. 27. Submits: FIT-BUT (2)

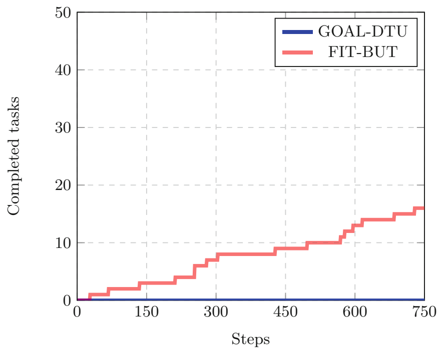

Fig. 28. Tasks: FIT-BUT (2)

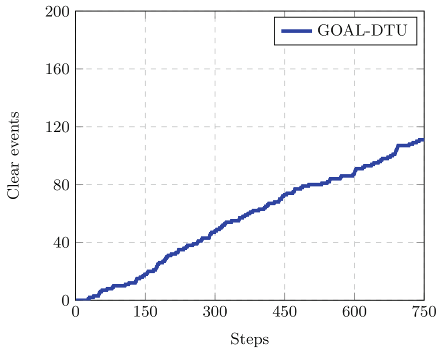

Fig. 29. Clear: FIT-BUT (2)

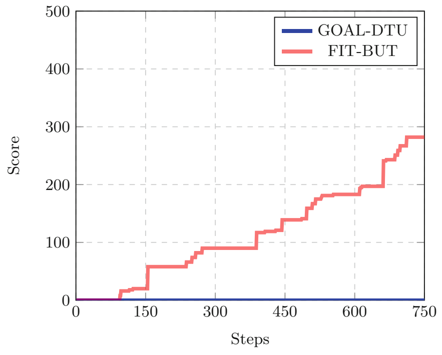

Fig. 30. Score: FIT-BUT (3)

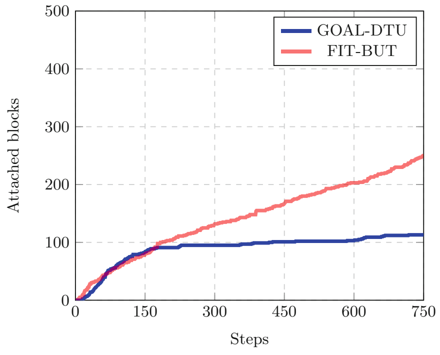

Fig. 31. Blocks: FIT-BUT (3)

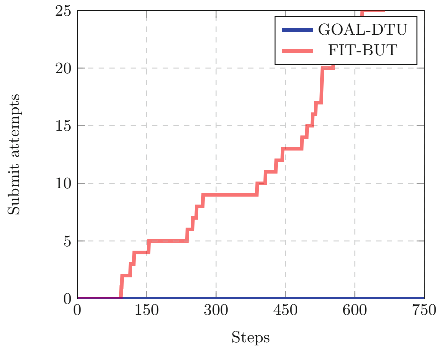

Fig. 32. Submits: FIT-BUT (3)

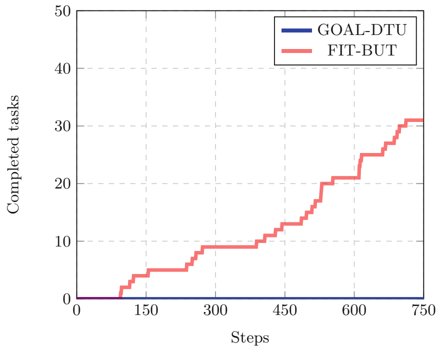

Fig. 33. Tasks: FIT-BUT (3)

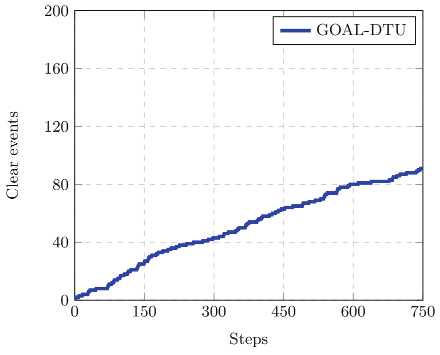

Fig. 34. Clear: FIT-BUT (3)

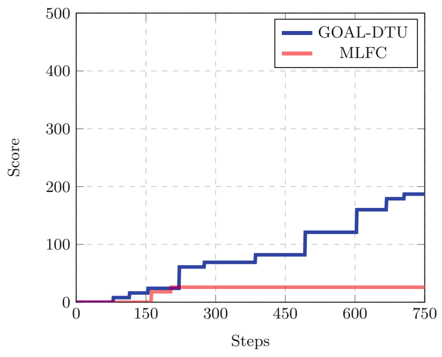

Fig. 35. Score: MLFC (1)

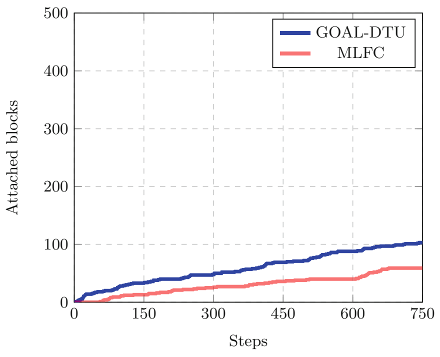

Fig. 36. Blocks: MLFC (1)

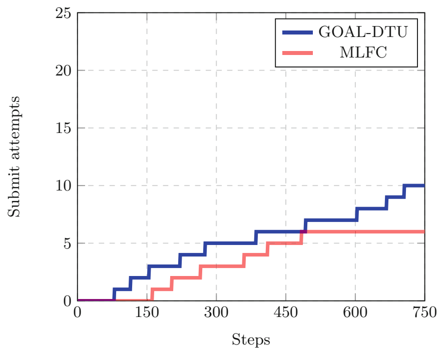

Fig. 37. Submits: MLFC (1)

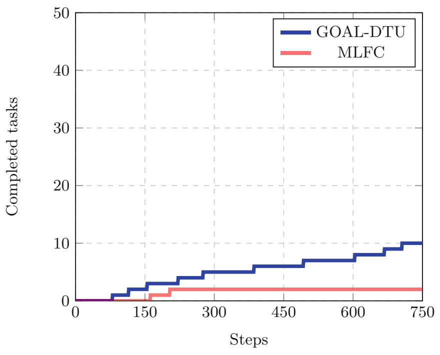

Fig. 38. Tasks: MLFC (1)

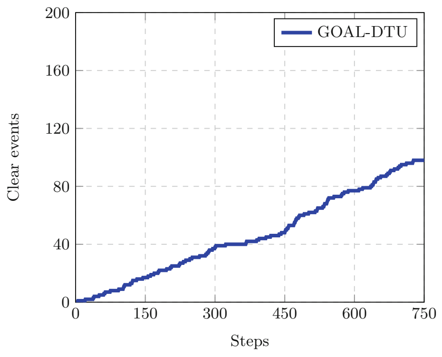

Fig. 39. Clear: MLFC (1)

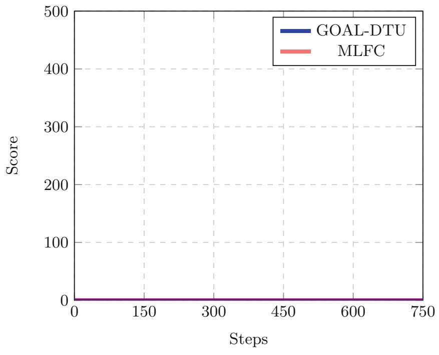

Fig. 40. Score: MLFC (2)

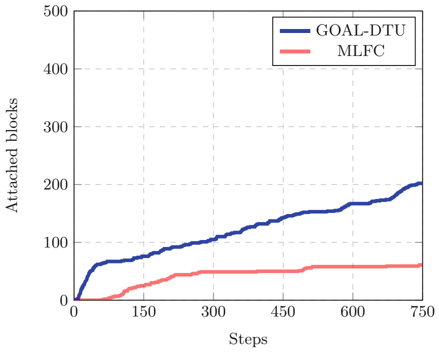

Fig. 41. Blocks: MLFC (2)

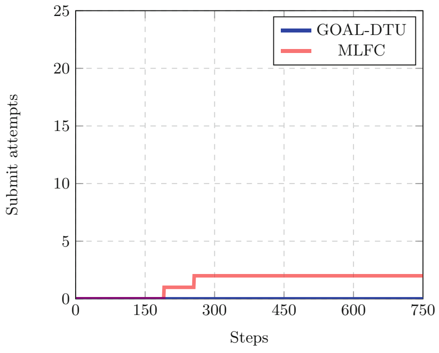

Fig. 42. Submits: MLFC (2)

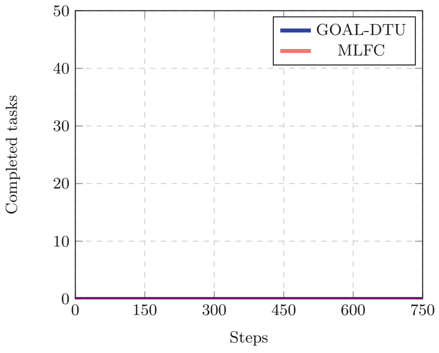

Fig. 43. Tasks: MLFC (2)

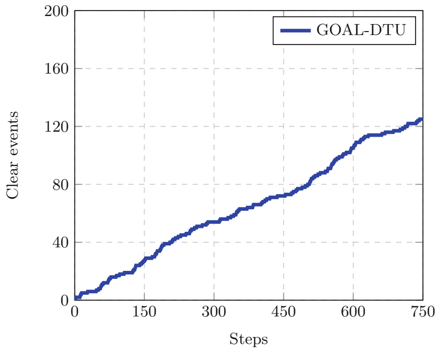

Fig. 44. Clear: MLFC (2)

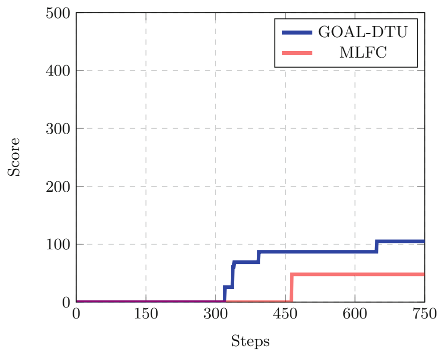

Fig. 45. Score: MLFC (3)

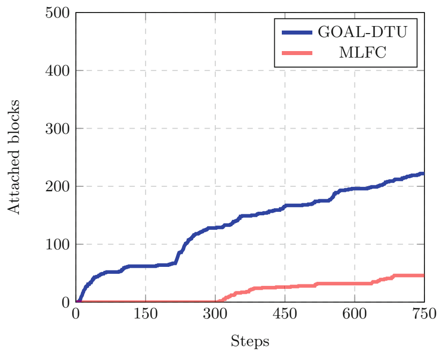

Fig. 46. Blocks: MLFC (3)

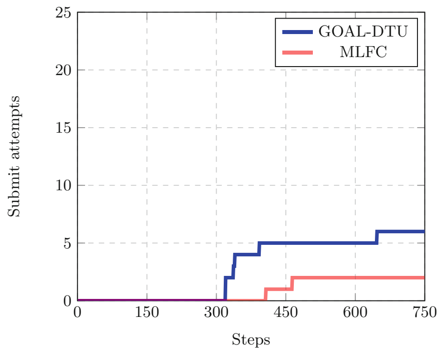

Fig. 47. Submits: MLFC (3)

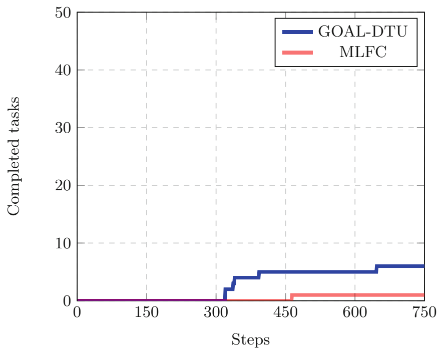

Fig. 48. Tasks: MLFC (3)

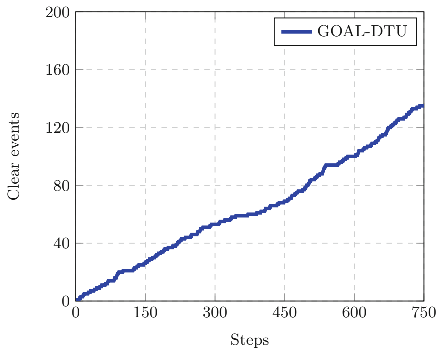

Fig. 49. Clear: MLFC (3)

Sadly, we experience our, by now common, total catastrophic failure in no fewer than two of the matches. We do not manage to solve a single task in either of the second or third simulations. Once our technical issues, errors, and bugs have been resolved, another match against FIT-BUT that would allow a more direct comparison of strategies would be interesting, as FIT-BUT seems to use a vastly different strategy than we do.

7.4 GOAL-DTU vs. JaCaMo Builders

GOAL-DTU had perfect conditions to get a high score in this match up because JaCaMo Builders seems to be using a defensive strategy without clearing GOAL-DTU agents. The low number of attached blocks is telling us that something went wrong for their agents in the two first simulations compared to the last simulation, see Figures Fig. 51, Fig. 56 and Fig. 61 (Figs. 52, 54, 57, 59, 60, 61, 62, 63 and 64).

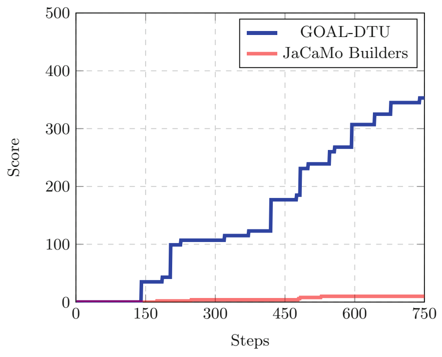

Fig. 50. Score: JaCaMo Builders (1)

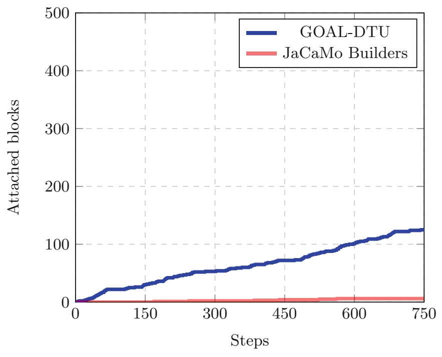

Fig. 51. Blocks: JaCaMo Builders (1)

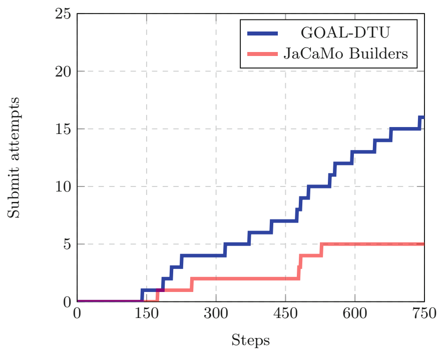

Fig. 52. Submits: JaCaMo Builders (1)

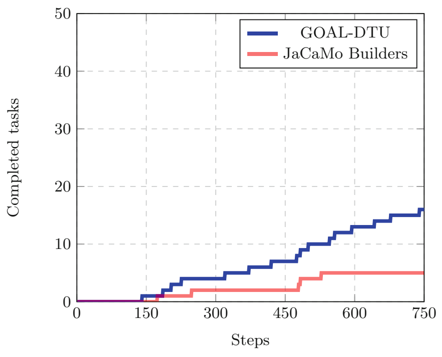

Fig. 53. Tasks: JaCaMo Builders (1)

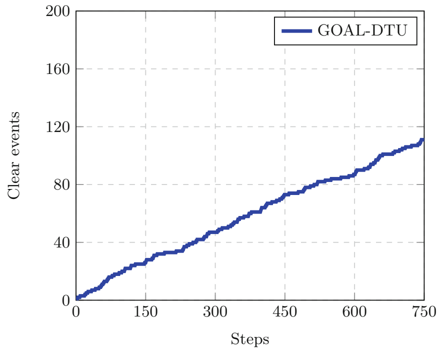

Fig. 54. Clear: JaCaMo Builders (1)

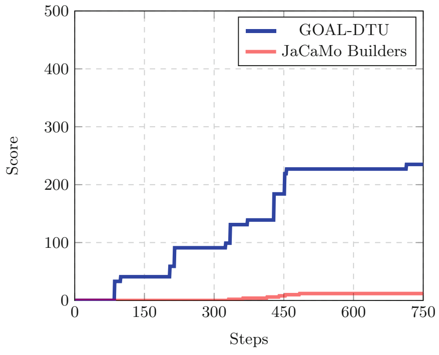

Fig. 55. Score: JaCaMo Builders (2)

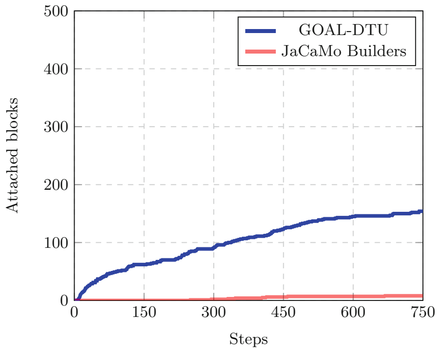

Fig. 56. Blocks: JaCaMo Builders (2)

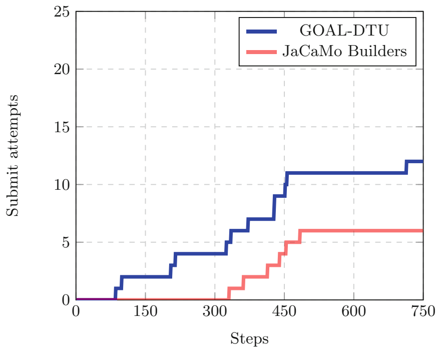

Fig. 57. Submits: JaCaMo Builders (2)

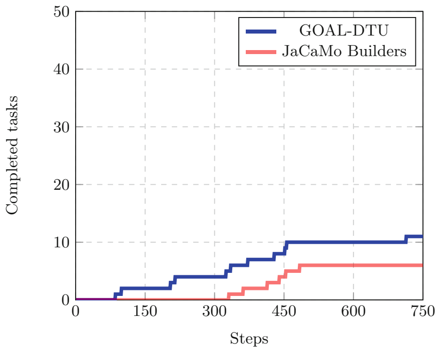

Fig. 58. Tasks: JaCaMo Builders (2)

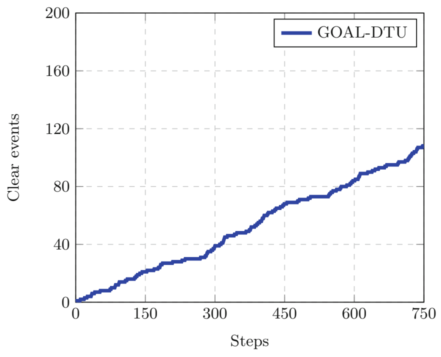

Fig. 59. Clear: JaCaMo Builders (2)

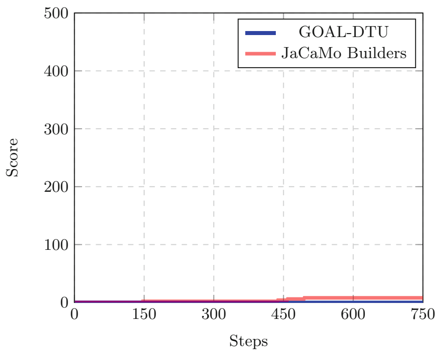

Fig. 60. Score: JaCaMo Builders (3)

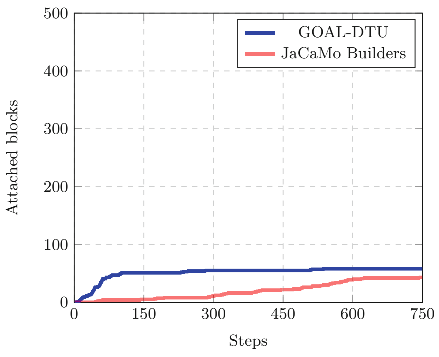

Fig. 61. Blocks: JaCaMo Builders (3)

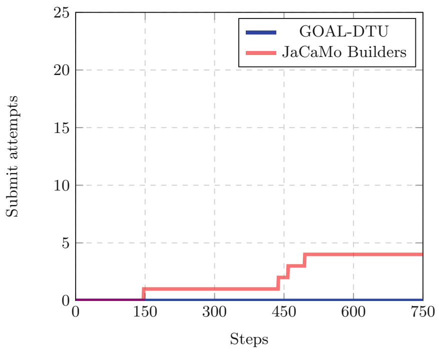

Fig. 62. Submits: JaCaMo Builders (3)

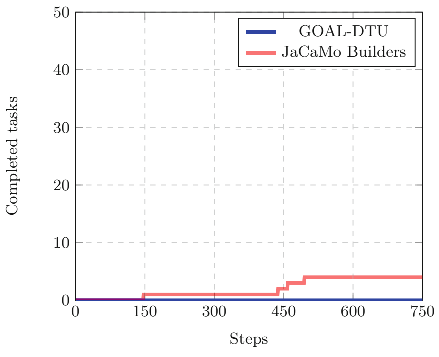

Fig. 63. Tasks: JaCaMo Builders (3)

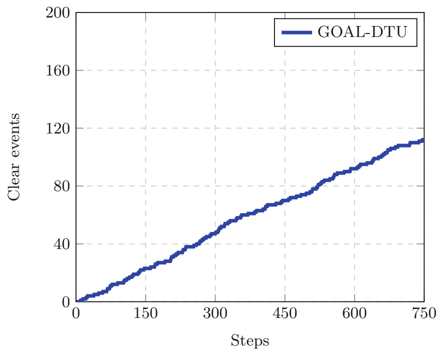

Fig. 64. Clear: JaCaMo Builders (3)

GOAL-DTU's prioritization of tasks combined with the A* path finding algorithm and static plans for tasks resulted in the highest score reached in the contest for all teams. Static plans work well in this case because agents of JaCaMo Builders didn't use much space in the goal zones and didn't clear GOAL-DTU agents in the goal zones.

GOAL-DTU received 22 points on average for the completed tasks in the first simulation, and 21 points on average in the second simulation, see Figures Fig. 50, Fig. 53, Fig. 55, Fig. 58. From step 456 until the end of the second simulation, GOAL-DTU only completes one task, see Fig. 58. It seems like the agents are out of sync and have wrong beliefs about the environment. This continues into the third simulation.

7.5 Free for All

GOAL-DTU is not well suited for simulations with many agents, especially not when the other agents are offensive and use clear actions. GOAL-DTU agents tried to defend themselves with clear actions while waiting in goal zones. The problem is that one agent can only defend against one other agent. If two agents are attacking the same GOAL-DTU agent, its only tactic for dodging clear actions is to move. It won't move due to the use of a static plan. This results in the agent being caught in the opposing agents' clear actions.

The victory in the first simulation seems a bit lucky. GOAL-DTU agents were attacked multiple times by agents from a second team while standing in goal zones. Agents from a third team appear before the attackers succeed. They try to clear the attackers such that the attackers have to move. The GOAL-DTU agents use this opening to submit the tasks.

These openings were less present in the second simulation with thirty agents. FIT-BUT assembles their patterns outside the goal zones such that agents are constantly moving. It's very hard to clear a moving agent, and FIT-BUT used that well to defend themselves.

8 Discussion

The following section sheds light on some of the issues we faced, both regarding system design, program bugs and technical issues. Lastly, we consider some of the further development work needed to improve our system.

8.1 System Robustness

During the contest, we experienced severe problems concerning robustness due to false information. When the problem did not occur, we generally achieved competitive scores. In fact, we won every match where we scored points. However, we identified occurrences of this problem in five out of twelve matches.

False information is detrimental to our system. The current version relies heavily on information being correct which is also the reason for our extensive connection-protocol between agents. The assumption that all agent offsets are correct allows the system to determine the dimensions of the map from the discrepancies of the agent offsets (as touched upon in Sect. 3). If an agent somehow incorrectly updates its location, this can lead to agents agreeing on incorrect dimensions of the map. In this case, no coordinates in the belief base will reflect their actual positions, and as a result the agents will be rendered useless. While this suggests that the system is not robust enough, one could also argue that, once false information is introduced into the system, we cannot expect coherent behaviour.

8.2 Technical Issues

We now shed light on some of the technical issues we faced when deploying our agent system.

Following the competition, we found a bug in the `eismassim` interface implementation. The bug caused agents to sometimes desynchronize with the server, ultimately hampering our agents' performance significantly. The root of the problem lies in the `eismassim` interface which connects the GOAL program to the server running the simulation: when the `eismassim` interface receives a step update from the server while still serving an agent response from the last step, the response will wrongfully be marked as responding to this updated step, and `eismassim` will then block the agent until the next request from the server. As a result, the `eismassim` interface will forward an outdated response to the server, and the agent is responding to a step *before* the agent has received the corresponding percepts from the server. In some cases, this synchronization error caused discrepancies between the believed location of the agent and its actual location, eventually leading the agents to infer the wrong map dimensions. Once an agent is stuck in this out-of-sync state, it cannot recover by itself.

We experienced a connection problem in cases where an agent was located on top of a task board or dispenser. While the agent would recognize this object, other agents would not perceive the object on which the agent was standing. This would potentially lead to non-matching connection requests, and in the worst case, wrongful connections. Ultimately, this would eventually paralyze the system and require a complete reboot.

8.3 Further Work

The technical issues regarding server synchronization and the rare connection problem have been fixed since the competition. However, there is still some general bug fixing left before the system works as intended. In particular, our implementation of a strategy using clearing was not working as intended during the competition and needs some attention. We consider it a prerequisite for more advanced improvements to fix most of these bugs.

We saw in the free-for-all simulations that our agents have a hard time assembling the patterns in the goal zone when a lot of other agents are present. This is due to the quite static approach our system takes when assembling tasks. An obvious improvement would be to give agents the freedom to change the point of assembly, without losing too much efficiency. This might allow the agents to assemble the pattern outside goal zones, and then simply have the submit agent move into the goal zone afterward.

Another observation we have made during the contest is that, especially in larger maps with lots of agents, we have too many idle agents. Once the map is explored and an agent has connected two blocks, the agent will simply roam the map waiting to be assigned a task. We have ideas that could lead to the idle agents being put to better use—for example by protecting goal zones.

Our system could be improved to better utilize a goal-driven approach, with a proper goal-hierarchy. For now, our agents only rely on very high-level goals, and the more intricate details are represented as beliefs. We intend to improve on this aspect in the future where agents have more hierarchical goals, i.e. high-level goals contain a number of sub-goals.

9 Conclusion

We have covered the main strategy of our agents. It is primarily based around proactive (and reactive) collection of blocks while exploring the map. Our agents reactively compose plans to solve available tasks that describe how patterns are to be aligned. While solving tasks, our agents will react to obstacles they meet along their way. When feasible, the agents employ the A* path finding algorithm to optimize short-term movement.

We have described how our agents acquire knowledge from the map, mainly as a result of an initial exploration strategy. We have further described how our agents communicate and utilize this knowledge to move around the map and complete tasks.

We have evaluated the performance of our system during the contest and have found that the system performs well for maps with few agents. As the number of agents increase, the system increasingly fails to assign all agents to tasks, and the assigned agents are less able to maneuver and assemble patterns inside the goal zones.

Finally, we have discussed ideas for general improvements to the system alongside a number of encountered issues. This includes both issues related to the system implementation but also regarding server connections.

In conclusion, we are satisfied with our placement for the contest and with the improvements we have made to the system compared to last year.

Acknowledgments. We thank Tobias Ahlbrecht, Asta Halkjær From and Benjamin Simon Stenbjerg Jepsen for discussions.

A Team Overview: Short Answers

A.1 Participants and Their Background

What was your motivation to participate in the contest? To work on implementing a multi-agent system capable of competing in a realistic, albeit simulated, scenario.

What is the history of your group? (course project, thesis, . . .) The name of our team is GOAL-DTU. We participated in the contest in 2009 and 2010 as the Jason-DTU team, in 2011 and 2012 as the Python-DTU team, in 2013 and 2014 as the GOAL-DTU team, in 2015/2016 as the Python-DTU team, in 2017 and 2018 as the Jason-DTU team and in 2019 as the GOAL-DTU team. We are affiliated with the Algorithms, Logic and Graphs section at DTU Compute, Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU). DTU Compute is located in the greater Copenhagen area. The main contact is associate professor Jørgen Villadsen, email: '`jovi@dtu.dk`'

What is your field of research? Which work therein is related? We are responsible for the Artificial Intelligence and Algorithms study line of the MSc in Computer Science and Engineering programme.

A.2 Statistics

Did you start your agent team from scratch or did you build on your own or someone else's agents (e.g. from last year)? We used as starting point our code from the MAPC 2019.

How much time did you invest in the contest (for programming, organizing your group, other)? We used approximately 160 h to qualify. From January until the contest we used approximately 300 h.

How was the time (roughly) distributed over the months before the contest? To qualify we used approximately 80 h in August and 80 h in September. In January we updated GOAL—the new version of GOAL was not compatible with most of the old code, thus a lot had to be rewritten. We also had to spend some time debugging GOAL itself. In February, the actual programming of the agents started.

How many lines of code did you produce for your final agent team? 2000 lines of code.

How many people were involved? 5 people: Jørgen Villadsen, Alexander Birch Jensen, Benjamin Simon Stenbjerg Jepsen, Erik Kristian Gylling and Jonas Weile.

When did you start working on your agents? We started working on our code from MAPC 2019 in August. As mentioned in the previous questions, large parts of the existing code had to be rewritten, however. This began in January.

A.3 Technology and Techniques

Did you make use of agent technology/AOSE methods or tools? What were your experiences?

Agent programming languages and/or frameworks? We used GOAL. We find that it is very intuitive and relatively easy for newcomers to learn which is an advantage as the programming team changes.

Methodologies (e.g. Prometheus)? No.

Notation (e.g. Agent UML)? No.

Coordination mechanisms (e.g. protocols, games, ...)? No.

Other (methods/concepts/tools)? We used the Eclipse IDE for programming (it has a GOAL add-on).

A.4 Agent System Details

How do your agents decide what to do? The agents reactively decide on their actions based on the current percepts, their beliefs and their goals.

How do your agents decide how to do it? By predetermined rules and actions.

How does the team work together? (i.e. coordination, information sharing, ...) How decentralised is your approach? The team communicates via messages and channels to share information and agree on plans. The approach is mostly decentralized, but certain planning tasks are currently delegated to a single agent at a time.

Do your agents make use of the following features: Planning, Learning, Organisations, Norms? If so, please elaborate briefly. The agents use planning to choose the tasks to pursue. A single agent is chosen to do the planning, but this agent relies on input from all other agents, and the planning agent is chosen dynamically at run time. The planning agent will search through assignment combinations and choose the most promising.

Can your agents change their general behavior during run time? If so, what triggers the changes? An agent will change its behaviour when it is chosen to take part in solving a task.

Did you have to make changes to the team (e.g. fix critical bugs) during the contest? We chose not to make changes during the contest.

How did you go about debugging your system? What kinds of measures could improve your debugging experience? We used log files to record the agents belief base and percepts. We experimented with linear temporal logic, but ultimately it did not make it to the final version.

During the contest you were not allowed to watch the matches. How did you understand what your team of agents was doing? By logging to the console. Admittedly, we could have done much more to improve this aspect.

Did you invest time in making your agents more robust/fault-tolerant? How? We spent some time on this, but not enough. This was one of our problems at the competition.

A.5 Scenario and Strategy

What is the main strategy of your agent team? First, to explore, have our agents find other agents, deduce the map dimensions and agree on a task planning agent. Once this agent has been found, it will continuously inquire the other agents about their available resources and try to create task plans. The task plan is sent to all agents involved in the plan, and these will try to solve it as efficiently as possible.

Please explain whether you think you came up with a good strategy or you rather enabled your agents to find the best strategy. We defined the strategy for our agents. Obviously, the agents have to find strategies for solving tasks and some aspects are only loosely defined.

Did you implement any strategy that tries to interfere with your opponents? We worked on some clearing strategies to defend goal cells, but they seemingly did more harm than good at the competition.

How do your agents decide which tasks to complete? Each task is ranked based on a simple heuristic based on the reward and the delivery time. The tasks are then checked based on their ranks in decreasing order, and the agents will try to complete any tasks they deem solvable.

How do your agents coordinate assembling and delivering a structure for a task? The agents create structured plans on how to assemble the structure. The plans are continuously checked to see if they remain feasible.

Which aspect(s) of the scenario did you find particularly challenging? It was a challenge that the map was a torus and also that the environment was dynamic.

A.6 And the Moral of it is ...

What did you learn from participating in the contest? We learned a lot about using GOAL to write multi-agent programs. We were reminded of the care it takes to develop and test in multi-agent environments.

What advice would you give to yourself before the contest/another team wanting to participate in the next? Start early, because unexpected problems will occur. Have a clear testing strategy.

What are the strong and weak points of your team? The coordination between agents is working quite well and the A* path finding helps agents to move directly. Agents could be more flexible in helping each other and prioritizing other agents' tasks over their own when it is better for the team.

Where did you benefit from your chosen programming language, methodology, tools, and algorithms? GOAL has built-in functionality that allows agents to communicate with one another and it has a predefined agent-cycle that is suitable for the belief-desire-intention model. A* was used by the agents to determine movement actions for short distances.

Which problems did you encounter because of your chosen technologies? We had problems with the EIS interface. These were most obvious during transitions between simulations. We also had some problems with GOAL and backwards compatibility.

Did you encounter previously unseen problems/bugs during the contest? We had a problem with our agents receiving false information and then not being able to do anything meaningful. This problem was not experienced beforehand—probably due to insufficient testing.

Did playing against other agent teams bring about new insights on your own agents? Yes, our agents are vulnerable to clear actions when they are waiting in the goal zones.

What would you improve (wrt. your agents) if you wanted to participate in the same contest a week from now (or next year)? If the contest was a week from now, we would mainly focus on bug fixing and thorough testing. If we had more time we would make better use of agents when they are not partaking in solving tasks. Also, we might look into some better defensive strategies and continuously revising plans to check if they could be optimized.

Which aspect of your team cost you the most time? We had major problems with a lot of the code not being compatible with the newest version of GOAL. Due to missing unit-tests, the problems were almost impossible to locate, and a lot of code had to be rewritten. This was a major setback. Furthermore, the A* algorithm used more CPU time than expected.

What can be improved regarding the contest/scenario for next year? As has already been suggested, running the agent programs on the server itself. If this was implemented, it would be interesting to decrease the time available for the agents to decide on their actions.

Why did your team perform as it did? Why did the other teams perform better/worse than you did? The A* and coordination between our agents made us fast at completing patterns. However, we had a large setback during January, which meant we had to rewrite most of the additions to the 2019 version, as well as spending some time on GOAL itself. This left little time for debugging. We thus found a lot of bugs during the competition.

If you participated in the "free-for-all" event after the contest, did you learn anything new about your agents from that? We had our suspicions confirmed—that the current strategy will be a lot less effective if there are many agents cluttering the goal zone. For such scenarios, we need a more dynamic task-solving approach.

References

1. Hindriks, K.V., Koeman, V.: The GOAL Agent Programming Language Home (2021). https://goalapl.atlassian.net/wiki
2. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming with declarative goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 228–243. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44631-1_16
3. Hindriks, K.V.: Programming rational agents in GOAL. In: El Fallah Seghrouchni, A., Dix, J., Dastani, M., Bordini, R.H. (eds.) Multi-Agent Programming, pp. 119–157. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-89299-3_4
4. Hindriks, K.V., Dix, J.: GOAL: a multi-agent programming language applied to an exploration game. In: Shehory, O., Sturm, A. (eds.) Agent-Oriented Software Engineering, pp. 235–258. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54432-3_12
5. Boss, N.S., Jensen, A.S., Villadsen, J.: Building multi-agent systems using Jason. Ann. Math. Artif. Intell. **59**, 373–388 (2010)
6. Vester, S., Boss, N.S., Jensen, A.S., Villadsen, J.: Improving multi-agent systems using Jason. Ann. Math. Artif. Intell. **61**, 297–307 (2011)
7. Ettienne, M.B., Vester, S., Villadsen, J.: Implementing a multi-agent system in python with an auction-based agreement approach. In: Dennis, L., Boissier, O., Bordini, R.H. (eds.) ProMAS 2011. LNCS (LNAI), vol. 7217, pp. 185–196. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31915-0_11

8. Villadsen, J., Jensen, A.S., Ettienne, M.B., Vester, S., Andersen, K.B., Frøsig, A.: Reimplementing a multi-agent system in Python. In: Dastani, M., Hübner, J.F., Logan, B. (eds.) ProMAS 2012. LNCS (LNAI), vol. 7837, pp. 205–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38700-5_13
9. Villadsen, J., et al.: Engineering a multi-agent system in GOAL. In: Cossentino, M., El Fallah Seghrouchni, A., Winikoff, M. (eds.) EMAS 2013. LNCS (LNAI), vol. 8245, pp. 329–338. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45343-4_18
10. Villadsen, J., From, A.H., Jacobi, S., Larsen, N.N.: Multi-agent programming contest 2016 - the Python-DTU team. Int. J. Agent-Oriented Softw. Eng. **6**(1), 86–100 (2018)
11. Villadsen, J., Fleckenstein, O., Hatteland, H., Larsen, J.B.: Engineering a multi-agent system in Jason and CArtAgO. Ann. Math. Artif. Intell. **84**, 57–74 (2018)
12. Villadsen, J., Bjørn, M.O., From, A.H., Henney, T.S., Larsen, J.B.: Multi-agent programming contest 2018—the Jason-DTU team. In: Ahlbrecht, T., Dix, J., Fiekas, N. (eds.) MAPC 2018. LNCS (LNAI), vol. 11957, pp. 41–71. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37959-9_3
13. Jensen, A.B., Villadsen, J.: GOAL-DTU: development of distributed intelligence for the multi-agent programming contest. In: Ahlbrecht, T., Dix, J., Fiekas, N., Krausburg, T. (eds.) MAPC 2019. LNCS (LNAI), vol. 12381, pp. 79–105. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59299-8_4